```
using StatsPlots, Plots, KernelDensity
```

```
fillviolin! (generic function with 1 method)
```

```
@shorthands fillviolin
```

```
violin_coords (generic function with 1 method)
```

```
function violin_coords(
    y;
    wts = nothing,
    trim::Bool = false,
    bandwidth = KernelDensity.default_bandwidth(y),
)
    kd =
        wts === nothing ? KernelDensity.kde(y, npoints = 200, bandwidth = bandwidth) :
        KernelDensity.kde(y, weights = weights(wts), npoints = 200, bandwidth =
        bandwidth)
    if trim
        xmin, xmax = Plots.ignorenan_extrema(y)
        inside = Bool[xmin <= x <= xmax for x in kd.x]
        return (kd.density[inside], kd.x[inside])
    end
    kd.density, kd.x
end
```

```
get_quantiles (generic function with 1 method)
```

```
get_quantiles(quantiles::AbstractVector) = quantiles
```

```
get_quantiles (generic function with 2 methods)
```

```
get_quantiles(x::Real) = [x]
```

```
get_quantiles (generic function with 3 methods)
```

```
get_quantiles(b::Bool) = b ? [0.5] : Float64[]
```

```
get_quantiles (generic function with 4 methods)
```

```
get_quantiles(n::Int) = range(0, 1, length = n + 2)[2:(end - 1)]
```

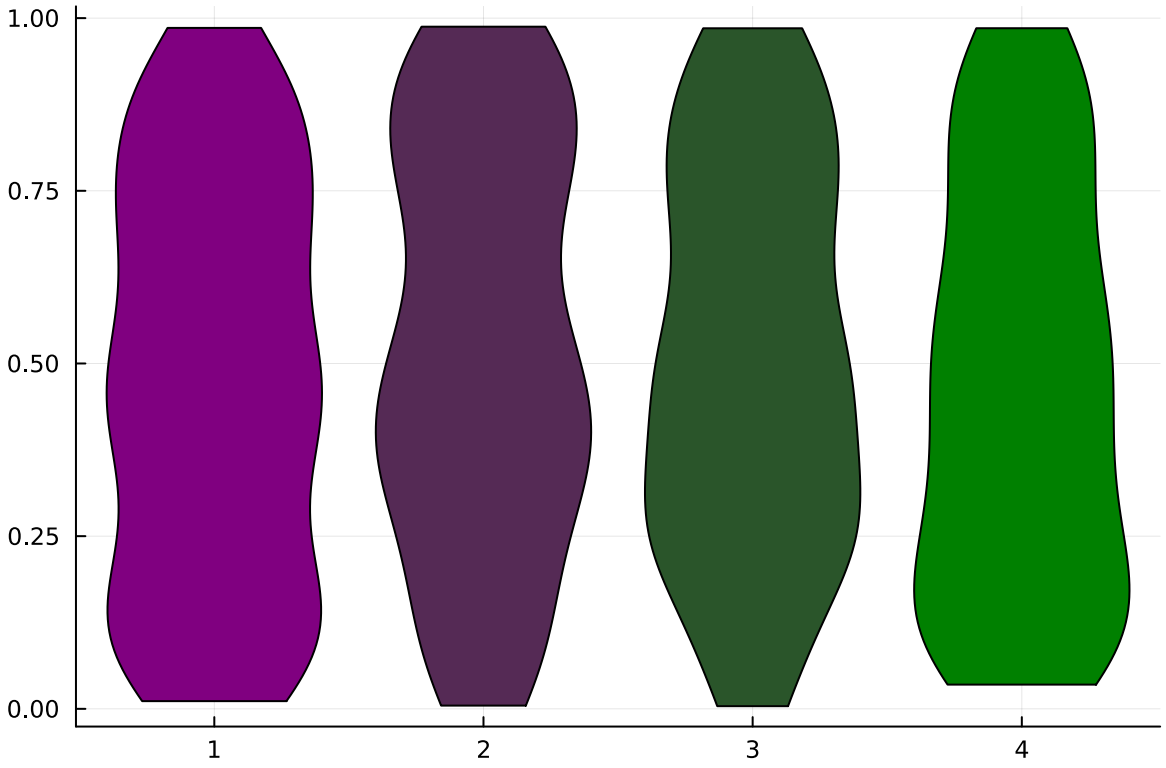
```
@recipe function f(
    ::Type{Val{:fillviolin}},
    x,
    y,
    z;
    trim = true,
    side = :both,
    show_mean = false,
    show_median = false,
    quantiles = Float64[],
    bandwidth = KernelDensity.default_bandwidth(y),
)
    # if only y is provided, then x will be UnitRange 1:size(y,2)
    if typeof(x) <: AbstractRange
        if step(x) == first(x) == 1
            x = plotattributes[:series_plotindex]
        else
            x = [getindex(x, plotattributes[:series_plotindex])]
        end
    end
    xsegs, ysegs = Segments(), Segments()
    qxsegs, qysegs = Segments(), Segments()
    mxsegs, mysegs = Segments(), Segments()
    glabels = sort(collect(unique(x)))

    bw = plotattributes[:bar_width]
    bw == nothing && (bw = 0.8)
    colors = plotattributes[:fillcolor]


    for (i, glabel) in enumerate(glabels)
        fy = y[filter(i -> Plots._cycle(x, i) == glabel, 1:length(y))]
        widths, centers = violin_coords(
            fy,
            trim = trim,
            wts = plotattributes[:weights],
            bandwidth = bandwidth,
        )
        isempty(widths) && continue

        # normalize
        hw = 0.5*Plots._cycle(bw, i)
        widths = hw * widths / Plots.ignorenan_maximum(widths)

        # make the violin
        xcenter = Plots.discrete_value!(plotattributes, :x, glabel)[1]
        xcoords = if (side === :right)
            vcat(widths, zeros(length(widths))) .+ xcenter
        elseif (side === :left)
            vcat(zeros(length(widths)), -reverse(widths)) .+ xcenter
        else
            vcat(widths, -reverse(widths)) .+ xcenter
        end
        ycoords = vcat(centers, reverse(centers))
```

```
55
56          push!(xsegs, xcoords)
57          push!(ysegs, ycoords)
58
59          if show_mean
60              mea = StatsBase.mean(fy)
61              mw = maximum(widths)
62              mx = xcenter .+ [-mw, mw] * 0.75
63              my = [mea, mea]
64              if side === :right
65                  mx[1] = xcenter
66              elseif side === :left
67                  mx[2] = xcenter
68              end
69
70              push!(mxsegs, mx)
71              push!(mysegs, my)
72          end
73
74          if show_median
75              med = StatsBase.median(fy)
76              mw = maximum(widths)
77              mx = xcenter .+ [-mw, mw] / 2
78              my = [med, med]
79              if side === :right
80                  mx[1] = xcenter
81              elseif side === :left
82                  mx[2] = xcenter
83              end
84
85              push!(qxsegs, mx)
86              push!(qysegs, my)
87          end
88
89          quantiles = get_quantiles(quantiles)
90          if !isempty(quantiles)
91              qy = quantile(fy, quantiles)
92              maxw = maximum(widths)
93
94              for i in eachindex(qy)
95                  qxi = xcenter .+ [-maxw, maxw] * (0.5 - abs(0.5 - quantiles[i]))
96                  qyi = [qy[i], qy[i]]
97                  if side === :right
98                      qxi[1] = xcenter
99                  elseif side === :left
100                     qxi[2] = xcenter
101                 end
102
103                 push!(qxsegs, qxi)
104                 push!(qysegs, qyi)
105             end
106
107             push!(qxsegs, [xcenter, xcenter])
108             push!(qysegs, [extrema(qy)...])
```

```
        end
    end

    @series begin
        seriestype := :shape
        x := xsegs.pts
        y := ysegs.pts
        fillcolor := popfirst!(colors.colors.colors)
        ()
    end

    if !isempty(mxsegs.pts)
        @series begin
            primary := false
            seriestype := :shape
            linestyle := :dot
            x := mxsegs.pts
            y := mysegs.pts
            ()
        end
    end

    if !isempty(qxsegs.pts)
        @series begin
            primary := false
            seriestype := :shape
            x := qxsegs.pts
            y := qysegs.pts
            ()
        end
    end

    seriestype := :shape
    primary := false
    x := []
    y := []
    ()
```

(:shape)

```
Plots.@deps fillviolin shape
```




```
1 fillviolin(rand(100,4), label=false, fillcolor=palette([:purple, :green], 4))
```